Введение

Иногда при большой кандидатуре на должность, работодателю приходится выбирать более подходящих людей. В данной работе мы проанализируем какие данные сотрудника больше всего влияют на его поведение, если сказать точнее, то при каких условиях работники чаще всего нарушают дисциплину. Целевая функция Disciplinary failure (yes=1; no=0) – дисциплинарная ошибка показывает, нарушал ли сотрудник дисциплину на работе. Остальные атрибуты: Информация об атрибутах:

1. Индивидуальная идентификация (ID)
2. Причина отсутствия (ICD).
3. Месяц отсутствия
4. День недели (понедельник (2), вторник (3), среда (4), четверг (5), пятница (6))
5. Сезоны (лето (1), осень (2), зима (3), весна (4))
6. Транспортные расходы
7. Расстояние от места жительства до работы (в километрах)
8. Время обслуживания
9. Возраст
10. Рабочая нагрузка Средняя / день
11. Попадание в цель
12. Дисциплинарная ошибка (да = 1; нет = 0) - целевая
13. Образование (средняя школа (1), выпускник (2), аспирант (3), магистр и доктор (4))
14. Сын (количество детей)
15. Социальный пьющий (да = 1; нет = 0)
16. Социальный курильщик (да = 1; нет = 0)
17. Домашнее животное (количество домашних животных)
18. Вес
19. Высота
20. Индекс массы тела
21. Время прогулов в часах (цель)

Датасет: https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work (https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work)

Основная часть

1.Импорт библиотек и загрузка датасета

In [2]:

```
import numpy as np
import pandas as pd
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))
from IPython.core.interactiveshell import InteractiveShell  #to run all statements in cell, not only the las
t
InteractiveShell.ast_node_interactivity = "all"
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from sklearn import preprocessing
from IPython.display import Image
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from xgboost import XGBClassifier
from sklearn.metrics import plot_confusion_matrix
import time
```

In [3]:

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

In [4]:

```
data = pd.read_excel('data/Absenteeism_at_work.xlsx')
data.head(10)
data.shape #Размер данных
data.isnull().sum() #Проверим нулевые значения
data.describe() # Основные статистические характеристки набора данных
data["Disciplinary failure"].unique() #Проверяем значения целевого признака
```

Out[4]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	...	Disciplinary failure	Education	Se
0	11	26	7	3	1	289	36	13	33	239554	...	0	1	
1	36	0	7	3	1	118	13	18	50	239554	...	1	1	
2	3	23	7	4	1	179	51	18	38	239554	...	0	1	
3	7	7	7	5	1	279	5	14	39	239554	...	0	1	
4	11	23	7	5	1	289	36	13	33	239554	...	0	1	
5	3	23	7	6	1	179	51	18	38	239554	...	0	1	
6	10	22	7	6	1	361	52	3	28	239554	...	0	1	
7	20	23	7	6	1	260	50	11	36	239554	...	0	1	
8	14	19	7	2	1	155	12	14	34	239554	...	0	1	
9	1	22	7	2	1	235	11	14	37	239554	...	0	3	

10 rows × 21 columns

Out[4]:

```
(740, 21)
```

Out[4]:

```
ID                                0
Reason for absence                0
Month of absence                  0
Day of the week                   0
Seasons                           0
Transportation expense            0
Distance from Residence to Work   0
Service time                      0
Age                               0
Work load Average/day             0
Hit target                        0
Disciplinary failure              0
Education                         0
Son                               0
Social drinker                    0
Social smoker                     0
Pet                               0
Weight                            0
Height                            0
Body mass index                   0
Absenteeism time in hours         0
dtype: int64
```

Out[4]:

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Av
count	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	74
mean	18.017568	19.216216	6.324324	3.914865	2.544595	221.329730	29.631081	12.554054	36.450000	27149
std	11.021247	8.433406	3.436287	1.421675	1.111831	66.952223	14.836788	4.384873	6.478772	3905
min	1.000000	0.000000	0.000000	2.000000	1.000000	118.000000	5.000000	1.000000	27.000000	20591
25%	9.000000	13.000000	3.000000	3.000000	2.000000	179.000000	16.000000	9.000000	31.000000	24438
50%	18.000000	23.000000	6.000000	4.000000	3.000000	225.000000	26.000000	13.000000	37.000000	26424
75%	28.000000	26.000000	9.000000	5.000000	4.000000	260.000000	50.000000	16.000000	40.000000	29421
max	36.000000	28.000000	12.000000	6.000000	4.000000	388.000000	52.000000	29.000000	58.000000	37888

8 rows × 21 columns

Out[4]:

```
array([0, 1], dtype=int64)
```

2.Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

In [5]:

```
data.dtypes
```

Out[5]:

```
ID                                int64
Reason for absence                int64
Month of absence                  int64
Day of the week                   int64
Seasons                           int64
Transportation expense            int64
Distance from Residence to Work   int64
Service time                      int64
Age                               int64
Work load Average/day             int64
Hit target                        int64
Disciplinary failure              int64
Education                         int64
Son                               int64
Social drinker                    int64
Social smoker                     int64
Pet                               int64
Weight                            int64
Height                            int64
Body mass index                   int64
Absenteeism time in hours         int64
dtype: object
```

In [6]:

```
data = data.drop(columns=['ID','Reason for absence','Month of absence','Day of the week',
       'Seasons', 'Work load Average/day ','Hit target','Education','Son','Pet','Social drinker',
       'Social smoker',
       'Absenteeism time in hours'],axis=1)
```

In [7]:

```
data.dtypes
```

Out[7]:

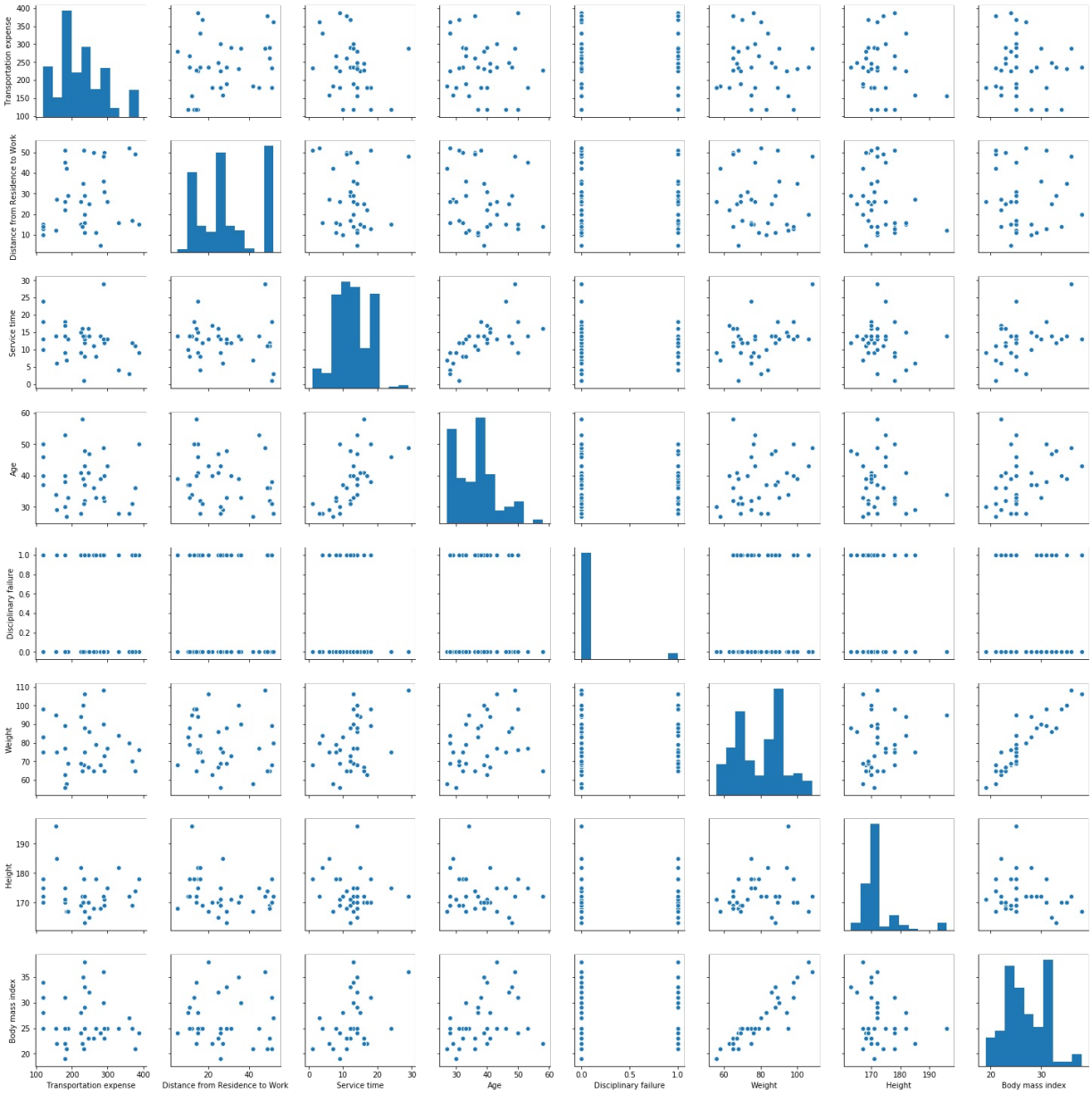
```
Transportation expense            int64
Distance from Residence to Work   int64
Service time                      int64
Age                               int64
Disciplinary failure              int64
Weight                            int64
Height                            int64
Body mass index                   int64
dtype: object
```

In [8]:

```
sns.pairplot(data)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x21a7cb55188>
```

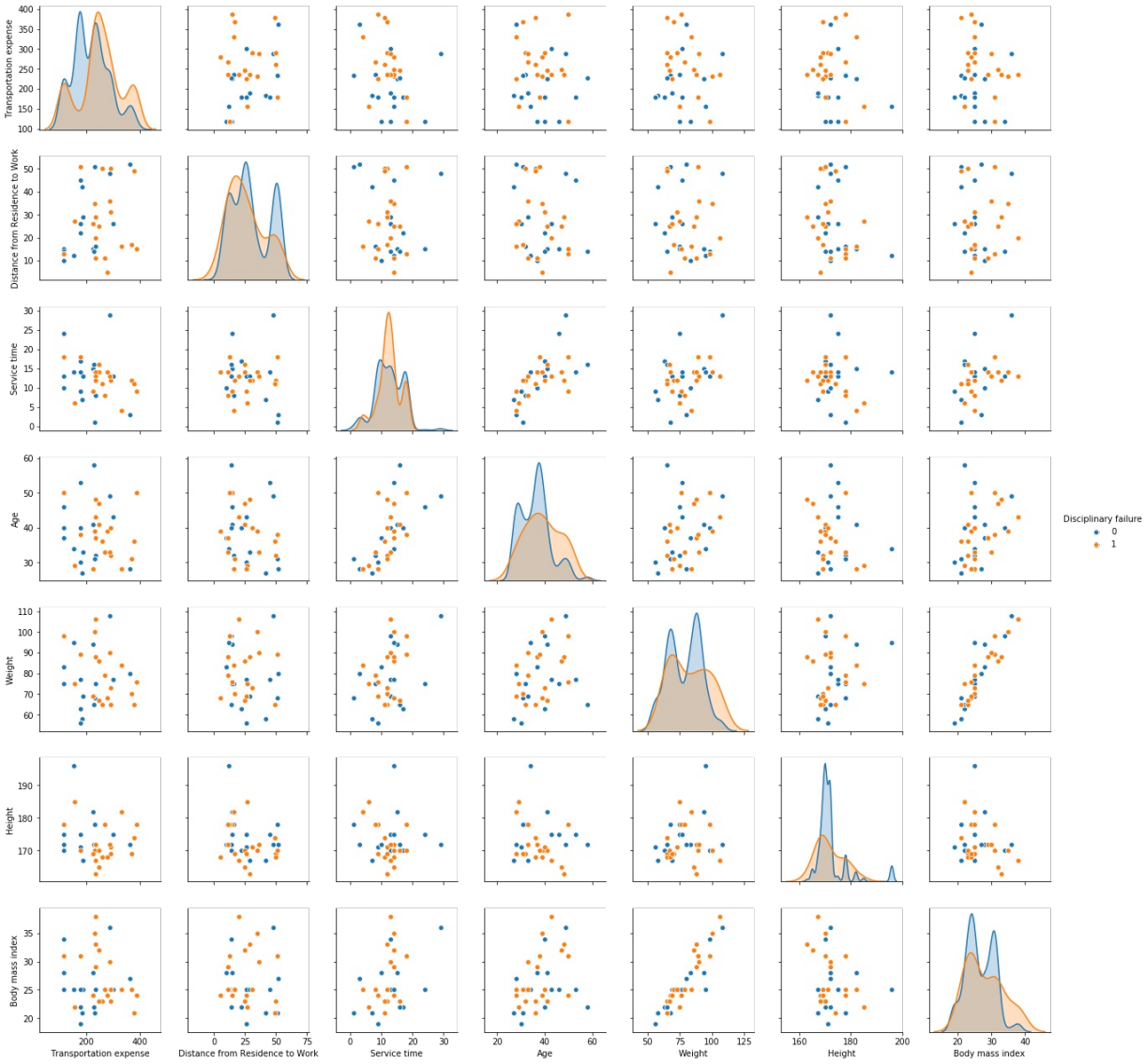


In [10]:

```
sns.pairplot(data, hue="Disciplinary failure")
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x21a7f754288>
```



In [11]:

```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['Disciplinary failure'].unique()
```

Out[11]:

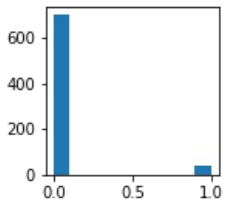
```
array([0, 1], dtype=int64)
```

In [12]:

```
# Оценим дисбаланс классов для Disciplinary failure
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['Disciplinary failure'])
plt.show()
```

Out[12]:

```
(array([700.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  40.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



In [13]:

```
data['Disciplinary failure'].value_counts()
```

Out[13]:

```
0    700
1     40
Name: Disciplinary failure, dtype: int64
```

In [15]:

```
total = data.shape[0]
class_0, class_1 = data['Disciplinary failure'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

```
Класс 0 составляет 94.59%, а класс 1 составляет 5.41%.
```

-Дисбаланс существенный

In [16]:

```
data.columns
```

Out[16]:

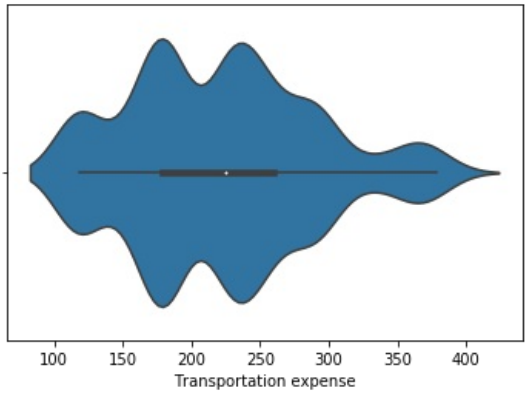
```
Index(['Transportation expense', 'Distance from Residence to Work',
       'Service time', 'Age', 'Disciplinary failure', 'Weight', 'Height',
       'Body mass index'],
      dtype='object')
```

In [18]:

```
# Скрипичные диаграммы для числовых колонок
for col in ['Transportation expense', 'Distance from Residence to Work',
       'Service time', 'Age', 'Disciplinary failure', 'Weight', 'Height',
       'Body mass index']:
    sns.violinplot(x=data[col])
    plt.show()
```

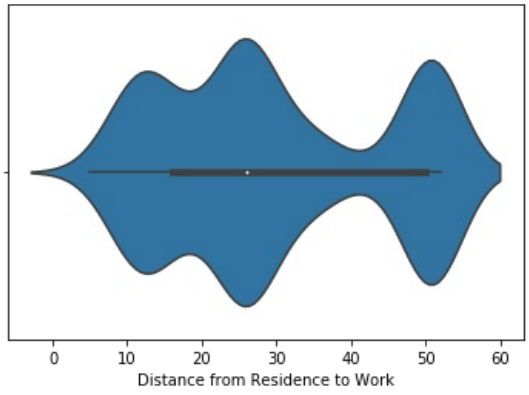
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a01fb3f88>
```



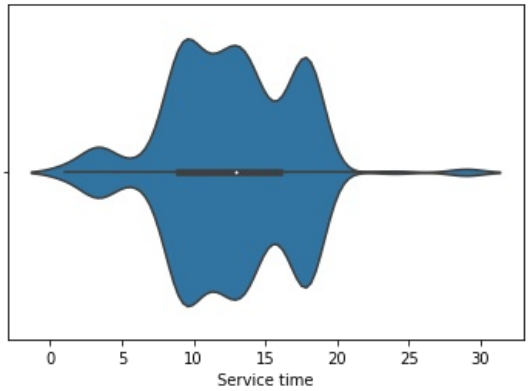
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a03221b88>
```



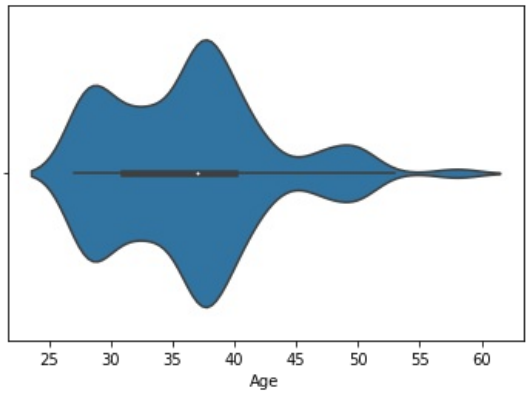
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a0424a6c8>
```



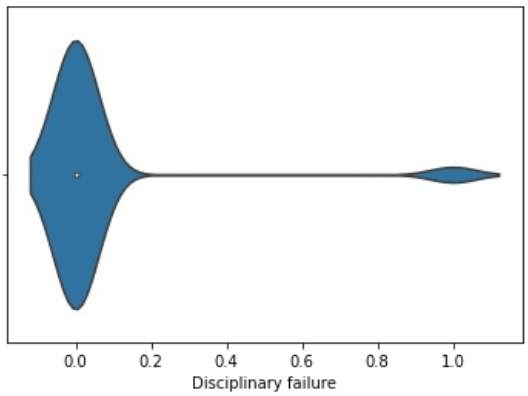
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a042b8708>
```



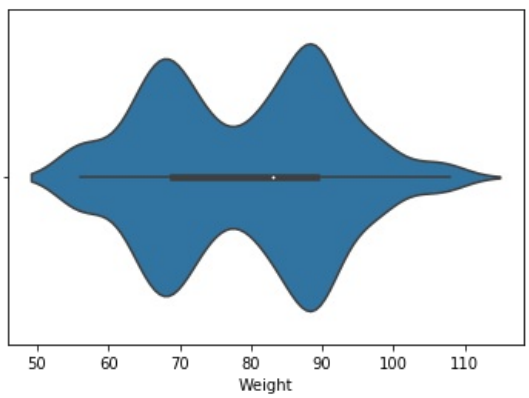
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a042fcf08>
```



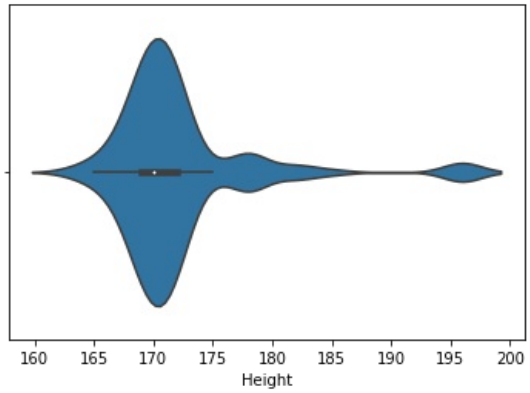
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a0424eb88>
```



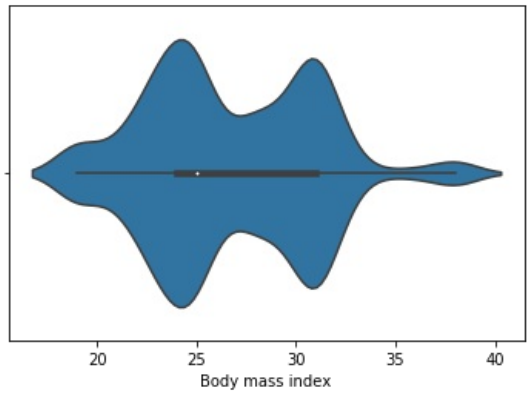
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a03a4de08>
```



Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a044121c8>
```



3.Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

In [19]:

```
scale_cols = ['Transportation expense', 'Distance from Residence to Work',
       'Service time', 'Age', 'Disciplinary failure', 'Weight', 'Height',
       'Body mass index']
```

In [20]:

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

In [21]:

```
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

In [22]:

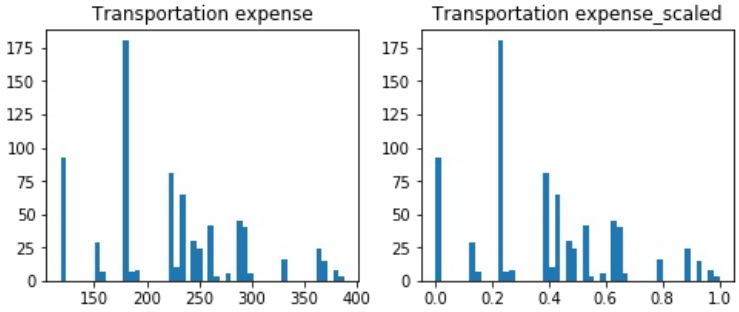
```
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

Out[22]:

```
(array([ 92.,   0.,   0.,   0.,   0.,   0.,  29.,   7.,   0.,   0.,   0.,
        180.,   7.,   8.,   0.,   0.,   0.,   0.,   0.,  81.,  10.,  65.,
          0.,  30.,  24.,   0.,  42.,   3.,   0.,   6.,   0.,  45.,  40.,
          5.,   0.,   0.,   0.,   0.,   0.,  16.,   0.,   0.,   0.,   0.,
          0.,  24.,  15.,   0.,   8.,   3.]),
 array([118. , 123.4, 128.8, 134.2, 139.6, 145. , 150.4, 155.8, 161.2,
        166.6, 172. , 177.4, 182.8, 188.2, 193.6, 199. , 204.4, 209.8,
        215.2, 220.6, 226. , 231.4, 236.8, 242.2, 247.6, 253. , 258.4,
        263.8, 269.2, 274.6, 280. , 285.4, 290.8, 296.2, 301.6, 307. ,
        312.4, 317.8, 323.2, 328.6, 334. , 339.4, 344.8, 350.2, 355.6,
        361. , 366.4, 371.8, 377.2, 382.6, 388. ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([ 92.,   0.,   0.,   0.,   0.,   0.,  29.,   7.,   0.,   0.,   0.,
        180.,   7.,   8.,   0.,   0.,   0.,   0.,   0.,  81.,  10.,  65.,
          0.,  30.,  24.,   0.,  42.,   3.,   0.,   6.,   0.,  45.,  40.,
          5.,   0.,   0.,   0.,   0.,   0.,  16.,   0.,   0.,   0.,   0.,
         24.,   0.,  15.,   0.,   8.,   3.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

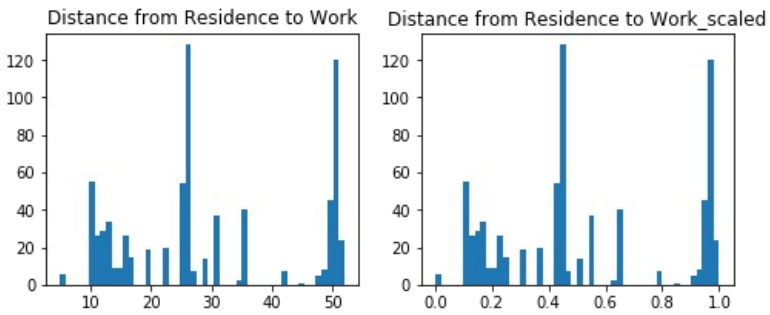


Out[22]:

```
(array([  6.,   0.,   0.,   0.,   0.,  55.,  26.,  29.,  34.,   9.,   9.,
         26.,  15.,   0.,   0.,  19.,   0.,   0.,  20.,   0.,   0.,  54.,
        128.,   7.,   0.,  14.,   0.,  37.,   0.,   0.,   0.,   2.,  40.,
          0.,   0.,   0.,   0.,   0.,   0.,   7.,   0.,   0.,   1.,   0.,
          0.,   5.,   8.,  45., 120.,  24.]),
 array([ 5.  ,  5.94,  6.88,  7.82,  8.76,  9.7 , 10.64, 11.58, 12.52,
        13.46, 14.4 , 15.34, 16.28, 17.22, 18.16, 19.1 , 20.04, 20.98,
        21.92, 22.86, 23.8 , 24.74, 25.68, 26.62, 27.56, 28.5 , 29.44,
        30.38, 31.32, 32.26, 33.2 , 34.14, 35.08, 36.02, 36.96, 37.9 ,
        38.84, 39.78, 40.72, 41.66, 42.6 , 43.54, 44.48, 45.42, 46.36,
        47.3 , 48.24, 49.18, 50.12, 51.06, 52.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([  6.,   0.,   0.,   0.,   0.,  55.,  26.,  29.,  34.,   9.,   9.,
         26.,  15.,   0.,   0.,  19.,   0.,   0.,  20.,   0.,   0.,  54.,
        128.,   7.,   0.,  14.,   0.,  37.,   0.,   0.,   0.,   2.,  40.,
          0.,   0.,   0.,   0.,   0.,   0.,   7.,   0.,   0.,   1.,   0.,
          0.,   5.,   8.,  45., 120.,  24.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

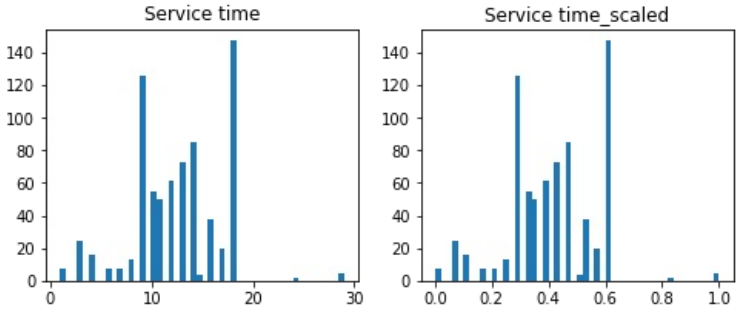


Out[22]:

```
(array([  7.,   0.,   0.,  24.,   0.,  16.,   0.,   0.,   7.,   0.,   7.,
          0.,  13.,   0., 126.,   0.,  55.,  50.,   0.,  61.,   0.,  73.,
          0.,  85.,   4.,   0.,  38.,   0.,  20.,   0., 147.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   5.]),
 array([ 1.  ,  1.56,  2.12,  2.68,  3.24,  3.8 ,  4.36,  4.92,  5.48,
         6.04,  6.6 ,  7.16,  7.72,  8.28,  8.84,  9.4 ,  9.96, 10.52,
        11.08, 11.64, 12.2 , 12.76, 13.32, 13.88, 14.44, 15.  , 15.56,
        16.12, 16.68, 17.24, 17.8 , 18.36, 18.92, 19.48, 20.04, 20.6 ,
        21.16, 21.72, 22.28, 22.84, 23.4 , 23.96, 24.52, 25.08, 25.64,
        26.2 , 26.76, 27.32, 27.88, 28.44, 29.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([  7.,   0.,   0.,  24.,   0.,  16.,   0.,   0.,   7.,   0.,   7.,
          0.,  13.,   0., 126.,   0.,  55.,  50.,   0.,  61.,   0.,  73.,
          0.,  85.,   0.,   4.,  38.,   0.,  20.,   0., 147.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   5.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

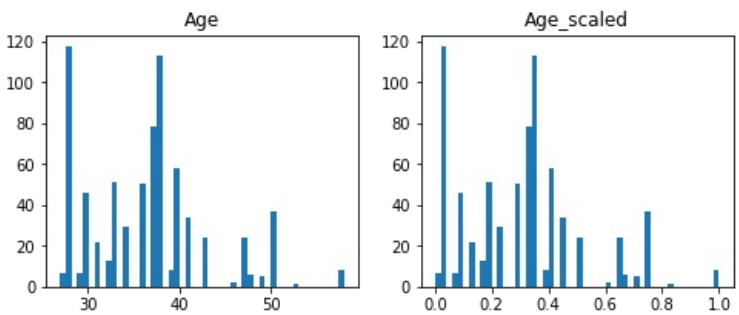


Out[22]:

```
(array([  7., 117.,   0.,   7.,  46.,   0.,  22.,   0.,  13.,  51.,   0.,
         29.,   0.,   0.,  50.,   0.,  78., 113.,   0.,   8.,  58.,   0.,
         34.,   0.,   0.,  24.,   0.,   0.,   0.,   0.,   2.,   0.,  24.,
          6.,   0.,   5.,   0.,  37.,   0.,   0.,   0.,   1.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   8.]),
 array([27.  , 27.62, 28.24, 28.86, 29.48, 30.1 , 30.72, 31.34, 31.96,
        32.58, 33.2 , 33.82, 34.44, 35.06, 35.68, 36.3 , 36.92, 37.54,
        38.16, 38.78, 39.4 , 40.02, 40.64, 41.26, 41.88, 42.5 , 43.12,
        43.74, 44.36, 44.98, 45.6 , 46.22, 46.84, 47.46, 48.08, 48.7 ,
        49.32, 49.94, 50.56, 51.18, 51.8 , 52.42, 53.04, 53.66, 54.28,
        54.9 , 55.52, 56.14, 56.76, 57.38, 58.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([  7., 117.,   0.,   7.,  46.,   0.,  22.,   0.,  13.,  51.,   0.,
         29.,   0.,   0.,  50.,   0.,  78., 113.,   0.,   8.,  58.,   0.,
         34.,   0.,   0.,  24.,   0.,   0.,   0.,   0.,   2.,   0.,  24.,
          6.,   0.,   5.,   0.,  37.,   0.,   0.,   0.,   1.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   8.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

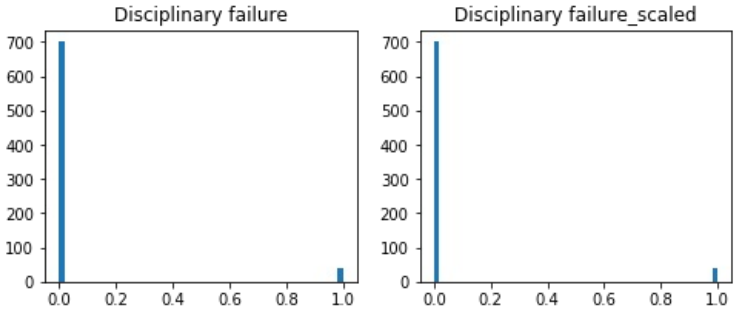


Out[22]:

```
(array([700.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,  40.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([700.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,  40.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

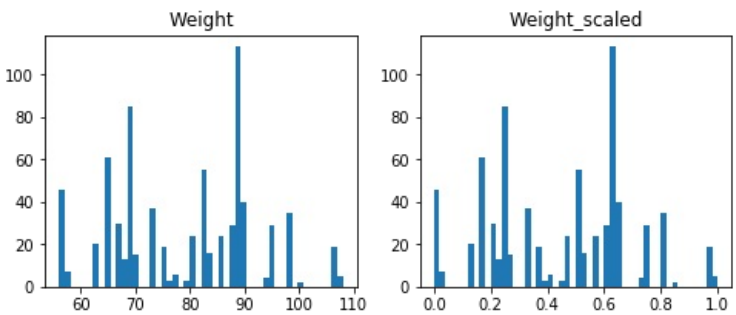


Out[22]:

```
(array([ 46.,   7.,   0.,   0.,   0.,   0.,  20.,   0.,  61.,   0.,  30.,
         13.,  85.,  15.,   0.,   0.,  37.,   0.,  19.,   3.,   6.,   0.,
          3.,  24.,   0.,  55.,  16.,   0.,  24.,   0.,  29., 113.,  40.,
          0.,   0.,   0.,   4.,  29.,   0.,   0.,  35.,   0.,   2.,   0.,
          0.,   0.,   0.,   0.,  19.,   5.]),
 array([ 56.  ,  57.04,  58.08,  59.12,  60.16,  61.2 ,  62.24,  63.28,
        64.32,  65.36,  66.4 ,  67.44,  68.48,  69.52,  70.56,  71.6 ,
        72.64,  73.68,  74.72,  75.76,  76.8 ,  77.84,  78.88,  79.92,
        80.96,  82.  ,  83.04,  84.08,  85.12,  86.16,  87.2 ,  88.24,
        89.28,  90.32,  91.36,  92.4 ,  93.44,  94.48,  95.52,  96.56,
        97.6 ,  98.64,  99.68, 100.72, 101.76, 102.8 , 103.84, 104.88,
       105.92, 106.96, 108.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([ 46.,   7.,   0.,   0.,   0.,   0.,  20.,   0.,  61.,   0.,  30.,
         13.,  85.,  15.,   0.,   0.,  37.,   0.,  19.,   3.,   6.,   0.,
          3.,  24.,   0.,  55.,  16.,   0.,  24.,   0.,  29., 113.,  40.,
          0.,   0.,   0.,   4.,  29.,   0.,   0.,  35.,   0.,   2.,   0.,
          0.,   0.,   0.,   0.,  19.,   5.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

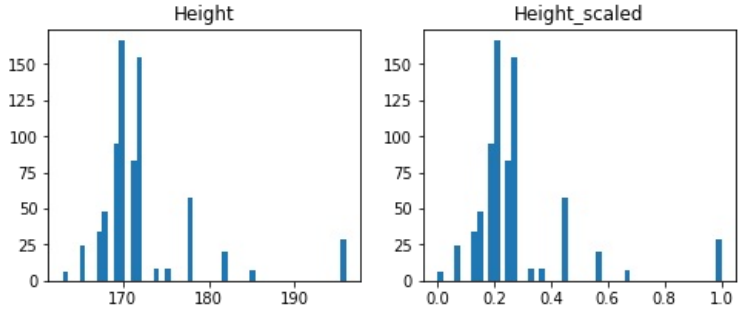


Out[22]:

```
(array([  6.,   0.,   0.,  24.,   0.,   0.,  34.,  48.,   0.,  95., 166.,
          0.,  83., 155.,   0.,   0.,   8.,   0.,   8.,   0.,   0.,   0.,
         57.,   0.,   0.,   0.,   0.,   0.,  20.,   0.,   0.,   0.,   0.,
          7.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,  29.]),
 array([163.  , 163.66, 164.32, 164.98, 165.64, 166.3 , 166.96, 167.62,
        168.28, 168.94, 169.6 , 170.26, 170.92, 171.58, 172.24, 172.9 ,
        173.56, 174.22, 174.88, 175.54, 176.2 , 176.86, 177.52, 178.18,
        178.84, 179.5 , 180.16, 180.82, 181.48, 182.14, 182.8 , 183.46,
        184.12, 184.78, 185.44, 186.1 , 186.76, 187.42, 188.08, 188.74,
        189.4 , 190.06, 190.72, 191.38, 192.04, 192.7 , 193.36, 194.02,
        194.68, 195.34, 196.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([  6.,   0.,   0.,  24.,   0.,   0.,  34.,  48.,   0.,  95., 166.,
          0.,  83., 155.,   0.,   0.,   8.,   0.,   8.,   0.,   0.,   0.,
         57.,   0.,   0.,   0.,   0.,   0.,  20.,   0.,   0.,   0.,   0.,
          7.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,  29.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```

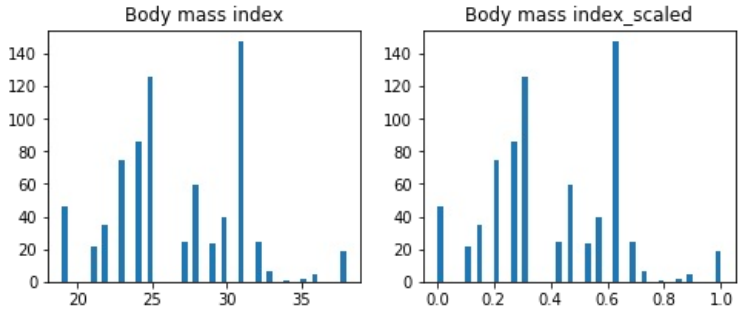


Out[22]:

```
(array([ 46.,   0.,   0.,   0.,   0.,  22.,   0.,  35.,   0.,   0.,  75.,
          0.,   0.,  86.,   0., 126.,   0.,   0.,   0.,   0.,   0.,  24.,
          0.,  59.,   0.,   0.,  23.,   0.,  40.,   0.,   0., 147.,   0.,
          0.,  24.,   0.,   6.,   0.,   0.,   1.,   0.,   0.,   2.,   0.,
          5.,   0.,   0.,   0.,   0.,  19.]),
 array([19.  , 19.38, 19.76, 20.14, 20.52, 20.9 , 21.28, 21.66, 22.04,
        22.42, 22.8 , 23.18, 23.56, 23.94, 24.32, 24.7 , 25.08, 25.46,
        25.84, 26.22, 26.6 , 26.98, 27.36, 27.74, 28.12, 28.5 , 28.88,
        29.26, 29.64, 30.02, 30.4 , 30.78, 31.16, 31.54, 31.92, 32.3 ,
        32.68, 33.06, 33.44, 33.82, 34.2 , 34.58, 34.96, 35.34, 35.72,
        36.1 , 36.48, 36.86, 37.24, 37.62, 38.  ]),
 <a list of 50 Patch objects>)
```

Out[22]:

```
(array([ 46.,   0.,   0.,   0.,   0.,  22.,   0.,  35.,   0.,   0.,  75.,
          0.,   0.,  86.,   0., 126.,   0.,   0.,   0.,   0.,   0.,  24.,
          0.,  59.,   0.,   0.,  23.,   0.,  40.,   0.,   0., 147.,   0.,
          0.,  24.,   0.,   6.,   0.,   0.,   1.,   0.,   0.,   2.,   0.,
          5.,   0.,   0.,   0.,   0.,  19.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <a list of 50 Patch objects>)
```



4.Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

In [77]:

```
corr_cols_1 = scale_cols + ['Disciplinary failure']
corr_cols_1
```

Out[77]:

```
['Transportation expense',
 'Distance from Residence to Work',
 'Service time',
 'Age',
 'Disciplinary failure',
 'Weight',
 'Height',
 'Body mass index',
 'Disciplinary failure']
```

In [78]:

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Disciplinary failure']
corr_cols_2
```

Out[78]:

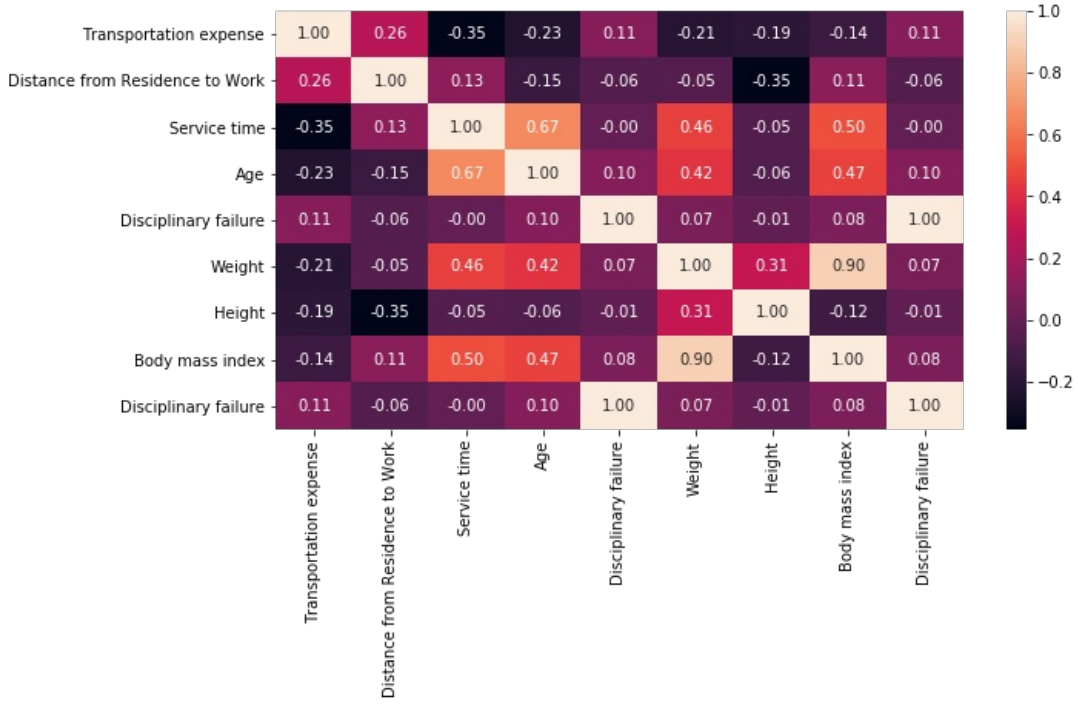
```
['Transportation expense_scaled',
 'Distance from Residence to Work_scaled',
 'Service time_scaled',
 'Age_scaled',
 'Disciplinary failure_scaled',
 'Weight_scaled',
 'Height_scaled',
 'Body mass index_scaled',
 'Disciplinary failure']
```

In [79]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[79]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a068ca1c8>
```

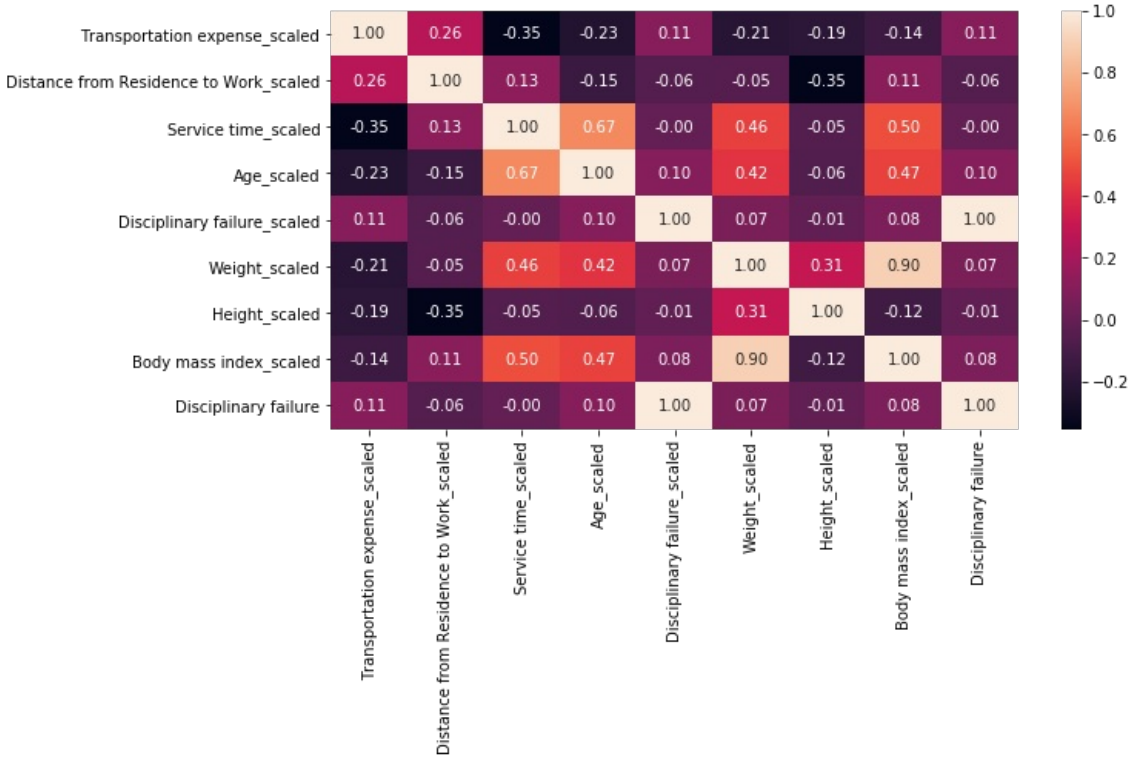


In [80]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21a04bc2dc8>
```



Вывод: Целевой признак не сильно коррелирует с осталными, но самые большие значения у Age и Transportation expense_scaled

In [4]:

```
plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
fig = data.Age.hist(bins=10)
fig.set_xlabel('Age')
fig.set_ylabel('Disciplinary failure')


plt.subplot(2, 2, 2)
fig = data.Weight.hist(bins=10)
fig.set_xlabel('Weight')
fig.set_ylabel('Disciplinary failure')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-1be3707b9f4f> in <module>
----> 1 plt.figure(figsize=(15,10))
      2
      3 plt.subplot(2, 2, 1)
      4 fig = data.Age.hist(bins=10)
      5 fig.set_xlabel('Age')

NameError: name 'plt' is not defined
```

5.Выбор метрик для последующей оценки качества моделей.

In [33]:

```
class Metrics:
    def __init__(self):
        self.df=pd.DataFrame(
            {'metric':pd.Series([], dtype='str'),
             'alg':pd.Series([], dtype='str'),
             'value':pd.Series([], dtype='float')})

    def setv(self, metric, alg, value):
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace=True)
        temp= [{'metric':metric, 'alg':alg, 'value':value}]
        self.df=self.df.append(temp, ignore_index=True)

    def getv(self, metric, ascending='True'):
        temp_data=self.df[self.df['metric']==metric]
        temp_data_2= temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5,5)):
        array_labels, array_metric = self.getv(metric,ascending)
        fix, ax1 = plt.subplots(figsize=figsize)
        pos=np.arange(len(array_metric))
        rects = ax1.barh (pos, array_metric,
                          align='center',
                          height=0.5,
                          tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)),color='white')
        plt.show()
```

6.Формирование обучающей и тестовой выборок на основе исходного набора данных.

In [39]:

```
features = data.drop(['Disciplinary failure'], axis=1)
target_tmp = data['Disciplinary failure']
target = pd.DataFrame({'Disciplinary failure':target_tmp.index, 'Disciplinary failure':target_tmp.values})
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.3, random_state=1)
X_train.shape
X_test.shape
Y_train.shape
Y_test.shape
```

Out[39]:

```
(518, 15)
```

Out[39]:

```
(222, 15)
```

Out[39]:

```
(518, 1)
```

Out[39]:

```
(222, 1)
```

7.Построение базового решения для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

Выберем методы:

KNeighbors

DecisionTreeClassifier

LinearSVC

XGBoost

Bagging

Данные методы являются относительно быстрыми, а так же помогают получить решение разными методами.

In [40]:

```
metric_logger = Metrics() #Запускаем класс логирования
def get_score(name,model,prediction):
    accuracy= accuracy_score(Y_test, prediction)
    precision= precision_score(Y_test, prediction)
    recall= recall_score(Y_test, prediction)
    print("accuracy_score: ", accuracy)
    print("precision_score: ", precision)
    print("recall_score: ", recall)
    metric_logger.setv("accuracy_score", name, accuracy)
    metric_logger.setv("precision_score", name, precision)
    metric_logger.setv("recall", name, recall)

def paint(model,prediction):
    draw_roc_curve(Y_test,prediction)
    plot_confusion_matrix(model, X_test, Y_test,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
```

In [41]:

```
cl0_0 = KNeighborsClassifier().fit(X_train, Y_train)
target0_0 = cl0_0.predict(X_test)
cl0_0
target0_0
```

Out[41]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

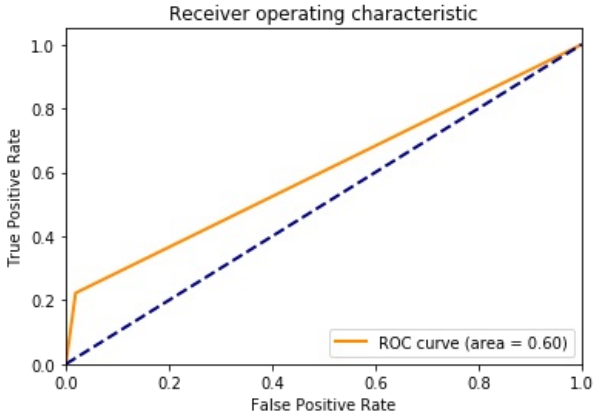
Out[41]:

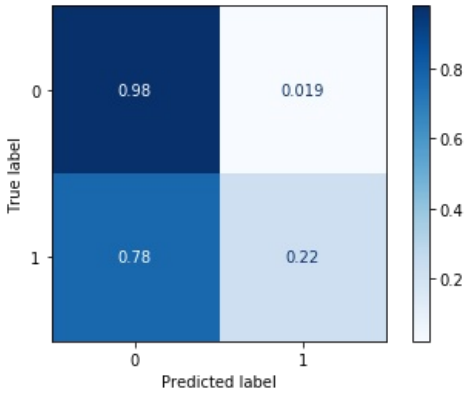
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [42]:

```
get_score('KNeighbors_base',cl0_0,target0_0)
paint(cl0_0,target0_0)
```

```
accuracy_score:  0.9504504504504504
precision_score:  0.3333333333333333
recall_score:  0.2222222222222222
```





In [43]:

```
cl1_0 = DecisionTreeClassifier(max_depth=None).fit(X_train, Y_train)
target1_0 = cl1_0.predict(X_test)
cl1_0
target1_0
```

Out[43]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

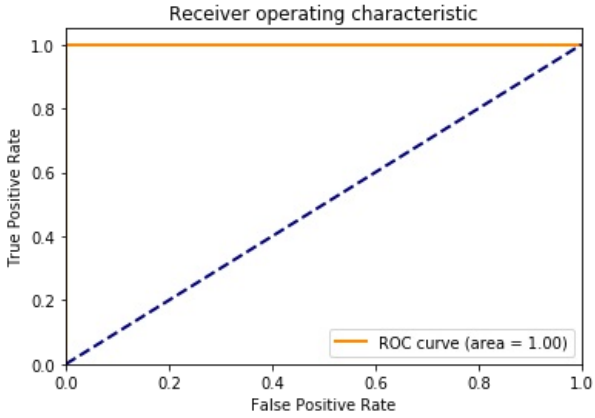
Out[43]:

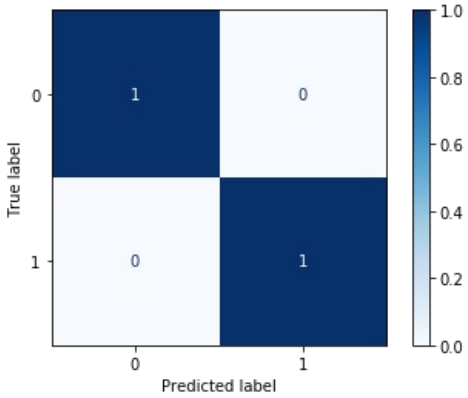
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0], dtype=int64)
```

In [44]:

```
get_score('Decision_base',cl1_0,target1_0)
paint(cl1_0,target1_0)
```

```
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





In [45]:

```
cl2_0 = LinearSVC(C=1.0, penalty='l1', dual=False, max_iter=10000, ).fit(X_train, Y_train)
target2_0 = cl2_0.predict(X_test)
cl2_0
target2_0
```

Out[45]:

```
LinearSVC(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=10000,
          multi_class='ovr', penalty='l1', random_state=None, tol=0.0001,
          verbose=0)
```

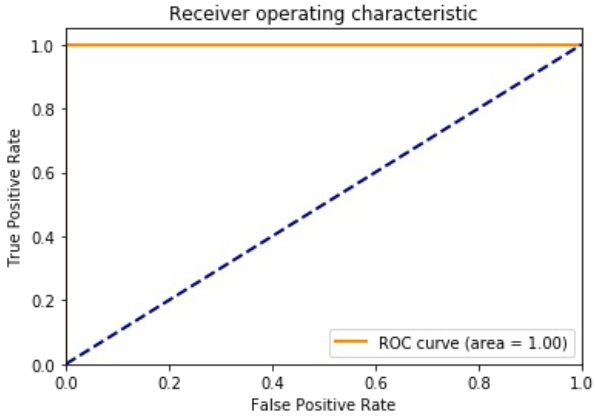
Out[45]:

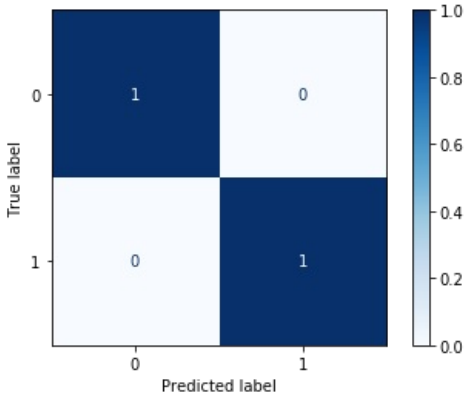
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0], dtype=int64)
```

In [46]:

```
get_score('Linear_base',cl2_0,target2_0)
paint(cl2_0,target2_0)
```

```
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





In [47]:

```
cl3_0 = XGBClassifier().fit(X_train, Y_train)
target3_0 = cl3_0.predict(X_test)
cl3_0
target3_0
```

Out[47]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

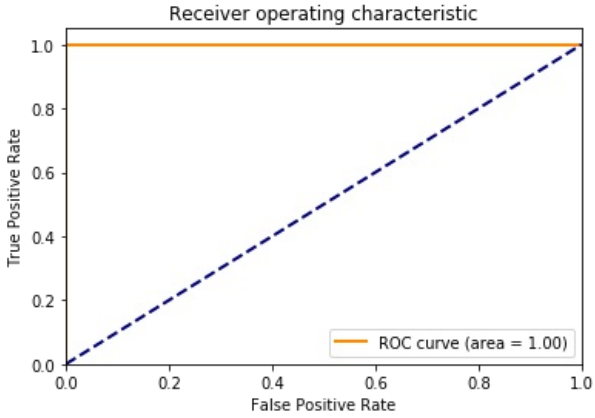
Out[47]:

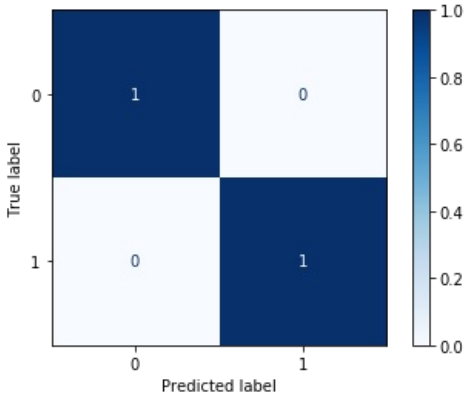
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0], dtype=int64)
```

In [48]:

```
get_score('XGBoost_base',cl3_0,target3_0)
paint(cl3_0,target3_0)
```

```
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





In [49]:

```
cl4_0 = BaggingClassifier().fit(X_train, Y_train)
target4_0 = cl4_0.predict(X_test)
cl4_0
target4_0
```

Out[49]:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=1.0, n_estimators=10,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

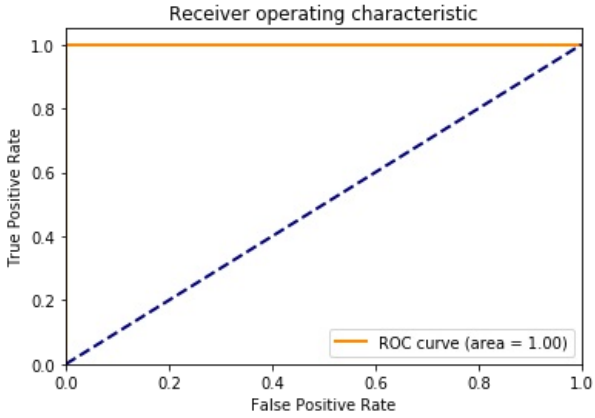
Out[49]:

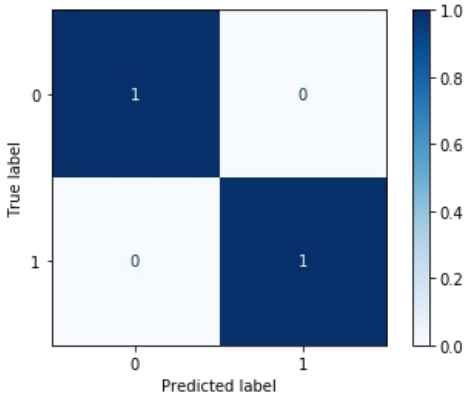
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0], dtype=int64)
```

In [50]:

```
get_score('Bagging_base',cl4_0,target4_0)
paint(cl4_0,target4_0)
```

```
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





8.Подбор гиперпараметров для выбранных моделей.

In [51]:

```
%%time

n_range = np.array(range(1,100,1))
tuned_parameters = [{'n_neighbors': n_range}]
print (tuned_parameters)
optimazer_0 = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv = 5).fit(X_train, Y_train)
print (optimazer_0.best_score_)
print (optimazer_0.best_params_)
```

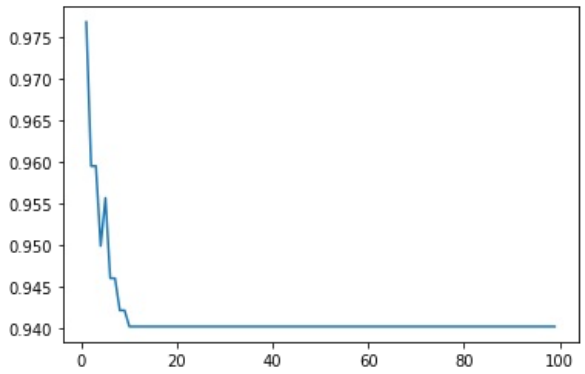
```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}]
0.9768110530246453
{'n_neighbors': 1}
Wall time: 9.25 s
```

In [52]:

```
y_range=np.array(optimazer_0.cv_results_['mean_test_score'])
plt.plot(n_range, y_range)
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x21a04a9c808>]
```



In [53]:

```
%%time

n_range1 = np.array(range(1,70,1))
tuned_parameters1 = [{'max_depth': n_range1}]
optimazer_1 = GridSearchCV(DecisionTreeClassifier(), tuned_parameters1, cv = 5).fit(X_train, Y_train)
print (optimazer_1.best_score_)
print (optimazer_1.best_params_)
```

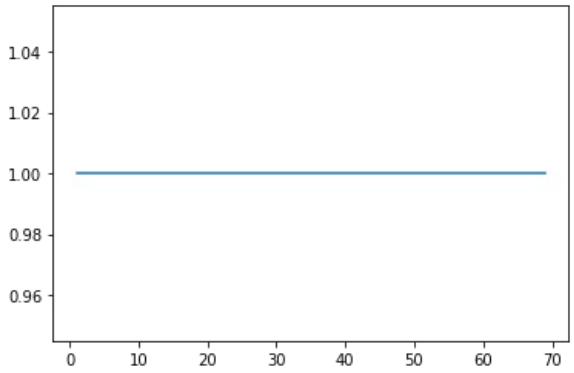
```
1.0
{'max_depth': 1}
Wall time: 2.66 s
```

In [54]:

```
y_range1=np.array(optimazer_1.cv_results_['mean_test_score'])
plt.plot(n_range1, y_range1)
```

Out[54]:

```
[<matplotlib.lines.Line2D at 0x21a04840688>]
```



In [55]:

```
%%time

n_range2 = np.arange(0.001, 1.5, 0.005)
tuned_parameters2 = [{'C': n_range2}]
#tuned_parameters2_1 = [{'tol': n_range2_1}]
optimazer_2 = GridSearchCV(LinearSVC(), tuned_parameters2, cv = 5).fit(X_train, Y_train)
print (optimazer_2.best_score_)
print (optimazer_2.best_params_)
```

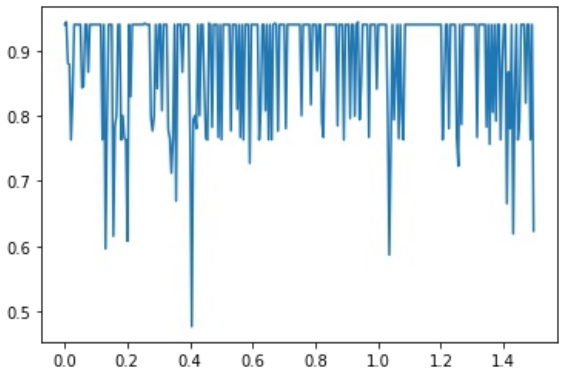
```
0.9440440627333831
{'C': 0.006}
Wall time: 1min 30s
```

In [56]:

```
y_range2=np.array(optimazer_2.cv_results_['mean_test_score'])
plt.plot(n_range2, y_range2)
```

Out[56]:

```
[<matplotlib.lines.Line2D at 0x21a06a61988>]
```



In [57]:

```
%%time


n_estimators = [50, 100, 150,200]
max_depth = [2, 4, 6, 8]
learning_rate = [0.01, 0.1, 0.2, 0.3]
tuned_parameters3 = dict(max_depth=max_depth, n_estimators=n_estimators, learning_rate=learning_rate)

optimazer_3 = GridSearchCV(XGBClassifier(), tuned_parameters3, cv = 5).fit(X_train, Y_train)

print (optimazer_3.best_score_)
print (optimazer_3.best_params_)
```

```
1.0
{'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 50}
Wall time: 22.6 s
```

In [58]:

```
optimazer_3.cv_results_
```

Out[58]:


```
{'mean_fit_time': array([0.04318781, 0.07500639, 0.1126092 , 0.12926755, 0.03018279,
        0.05023708, 0.06483278, 0.0837759 , 0.03002167, 0.05335884,
        0.07619667, 0.10631509, 0.03550568, 0.04707408, 0.06235666,
        0.07955966, 0.02912111, 0.04328537, 0.05884581, 0.07377396,
        0.03031921, 0.04391408, 0.06842299, 0.08846583, 0.04141073,
        0.06382904, 0.06273508, 0.07929049, 0.03181782, 0.0550529 ,
        0.07317314, 0.09614668, 0.03549705, 0.06443524, 0.06251001,
        0.07179942, 0.02963161, 0.04777679, 0.06619649, 0.07711034,
        0.03161001, 0.04761658, 0.06208887, 0.07859783, 0.03230791,
        0.04647574, 0.06233683, 0.10123153, 0.03859968, 0.06901479,
        0.09214363, 0.1068789 , 0.04338441, 0.06298656, 0.07328553,
        0.07696109, 0.03122692, 0.05365572, 0.06263804, 0.10511832,
        0.03470697, 0.05664868, 0.07336783, 0.0880796 ]),
 'std_fit_time': array([0.00792973, 0.00433067, 0.00650016, 0.00704509, 0.0022234 ,
        0.0029509 , 0.00335578, 0.00260122, 0.00155461, 0.00274937,
        0.01030022, 0.01365417, 0.00360104, 0.0014666 , 0.00114349,
        0.00234271, 0.00159571, 0.00101721, 0.00089555, 0.0050532 ,
        0.00325275, 0.00092879, 0.00842719, 0.01173781, 0.00576853,
        0.01011188, 0.0048267 , 0.00467632, 0.00296372, 0.00452173,
        0.00714734, 0.01170383, 0.00357604, 0.01506075, 0.00469991,
        0.004839  , 0.00246482, 0.00232775, 0.00636779, 0.00325034,
        0.00283565, 0.00360448, 0.00367853, 0.00536161, 0.00203625,
        0.00299913, 0.00305646, 0.01506819, 0.00255992, 0.00677572,
        0.00808024, 0.01018967, 0.00305066, 0.00503694, 0.01440797,
        0.00671728, 0.00172935, 0.00837662, 0.00460058, 0.02790101,
        0.00635145, 0.01094962, 0.01256346, 0.00262515]),
 'mean_score_time': array([0.0061832 , 0.00578427, 0.00618386, 0.00738173, 0.00458994,
        0.00458889, 0.00538621, 0.00526028, 0.00508962, 0.00558634,
        0.00578599, 0.00598497, 0.00538654, 0.00518699, 0.00501366,
        0.00538626, 0.00578547, 0.00558577, 0.00570326, 0.00538816,
        0.00516238, 0.00578585, 0.00658188, 0.00558629, 0.00678287,
        0.00618391, 0.00499048, 0.00548978, 0.00598469, 0.00469151,
        0.00536532, 0.00569253, 0.00579596, 0.00696292, 0.00600567,
        0.005198  , 0.00558624, 0.00580969, 0.00581255, 0.00509677,
        0.00478759, 0.00559254, 0.00551791, 0.00560656, 0.00499563,
```

```
       0.00534129, 0.00558472, 0.00598502, 0.00642128, 0.00698228,

       0.00619392, 0.00599494, 0.00558562, 0.0073885 , 0.00529246,
       0.0051877 , 0.00618386, 0.00558548, 0.00589213, 0.00618472,
       0.00718207, 0.00638304, 0.00576277, 0.00585003]),
 'std_score_time': array([9.76981266e-04, 7.47054340e-04, 9.76698277e-04, 1.35284564e-03,
       7.95861896e-04, 4.88461530e-04, 4.89415299e-04, 3.89078972e-04,
       4.86228973e-04, 7.98082986e-04, 7.47388748e-04, 6.30826481e-04,
       4.88558366e-04, 7.46748478e-04, 6.31973522e-04, 4.87916818e-04,
       9.76649377e-04, 4.88889411e-04, 8.51169639e-04, 4.90843432e-04,
       7.56067570e-04, 3.98827226e-04, 1.35239548e-03, 1.35184741e-03,
       1.59562874e-03, 1.46623313e-03, 6.33924786e-04, 4.46641543e-04,
       1.26172861e-03, 3.97464255e-04, 1.31375987e-03, 8.75185125e-04,
       1.32446228e-03, 1.26172603e-03, 9.21717181e-04, 7.58957620e-04,
       4.88890830e-04, 7.51671456e-04, 1.61386083e-03, 6.78378892e-04,
       7.45244668e-04, 8.10639106e-04, 1.03098576e-03, 1.21731094e-03,
       1.52456704e-05, 4.38969772e-04, 4.88812513e-04, 8.91591057e-04,
       8.57527610e-04, 1.40976168e-03, 4.20308691e-04, 6.30492270e-04,
       4.89065548e-04, 8.04544060e-04, 4.02860901e-04, 3.98421584e-04,
       1.16286776e-03, 1.19743407e-03, 9.25994528e-04, 7.47015862e-04,
       2.55432474e-03, 1.19694937e-03, 2.08402177e-03, 1.10691383e-03]),
 'param_learning_rate': masked_array(data=[0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.0
1,
                   0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.1,
                   0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
                   0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2,
                   0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3, 0.3,
                   0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
                   0.3, 0.3],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 8, 2, 2,
                   2, 2, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 8, 2, 2, 2, 2,
                   4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 8, 2, 2, 2, 2, 4, 4,
                   4, 4, 6, 6, 6, 6, 8, 8, 8, 8],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[50, 100, 150, 200, 50, 100, 150, 200, 50, 100, 150,
                   200, 50, 100, 150, 200, 50, 100, 150, 200, 50, 100,
                   150, 200, 50, 100, 150, 200, 50, 100, 150, 200, 50,
                   100, 150, 200, 50, 100, 150, 200, 50, 100, 150, 200,
                   50, 100, 150, 200, 50, 100, 150, 200, 50, 100, 150,
                   200, 50, 100, 150, 200, 50, 100, 150, 200],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 50},
  {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 150},
  {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 200},
  {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 50},
  {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 100},
  {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 150},
  {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 200},
  {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 50},
  {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100},
  {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 150},
  {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 200},
  {'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 50},
  {'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 100},
  {'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 150},
```

```
  {'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 200},

  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 50},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 150},
  {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 200},
  {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 50},
  {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 150},
  {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200},
  {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 50},
  {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 150},
  {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200},
  {'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 50},
  {'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 100},
  {'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 150},
  {'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 200},
  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 50},
  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 150},
  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 200},
  {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 50},
  {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 100},
  {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 150},
  {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 200},
  {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 50},
  {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 100},
  {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 150},
  {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 200},
  {'learning_rate': 0.2, 'max_depth': 8, 'n_estimators': 50},
  {'learning_rate': 0.2, 'max_depth': 8, 'n_estimators': 100},
  {'learning_rate': 0.2, 'max_depth': 8, 'n_estimators': 150},
  {'learning_rate': 0.2, 'max_depth': 8, 'n_estimators': 200},
  {'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 50},
  {'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 100},
  {'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 150},
  {'learning_rate': 0.3, 'max_depth': 2, 'n_estimators': 200},
  {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 50},
  {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 100},
  {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 150},
  {'learning_rate': 0.3, 'max_depth': 4, 'n_estimators': 200},
  {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 50},
  {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 100},
  {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 150},
  {'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 200},
  {'learning_rate': 0.3, 'max_depth': 8, 'n_estimators': 50},
  {'learning_rate': 0.3, 'max_depth': 8, 'n_estimators': 100},
  {'learning_rate': 0.3, 'max_depth': 8, 'n_estimators': 150},
  {'learning_rate': 0.3, 'max_depth': 8, 'n_estimators': 200}],
 'split0_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'split1_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'split2_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'split3_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1
.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'mean_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'std_test_score': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 'rank_test_score': array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])}
```

In [59]:

```
%%time
n_estimators4 = [20, 50, 100, 150, 200, 250]
tuned_parameters4 = [{'n_estimators': n_estimators4}]
optimazer_4 = GridSearchCV(BaggingClassifier(), tuned_parameters4, cv = 5).fit(X_train, Y_train)
print (optimazer_4.best_score_)
print (optimazer_4.best_params_)
```

```
1.0
{'n_estimators': 20}
Wall time: 13.8 s
```

In [60]:

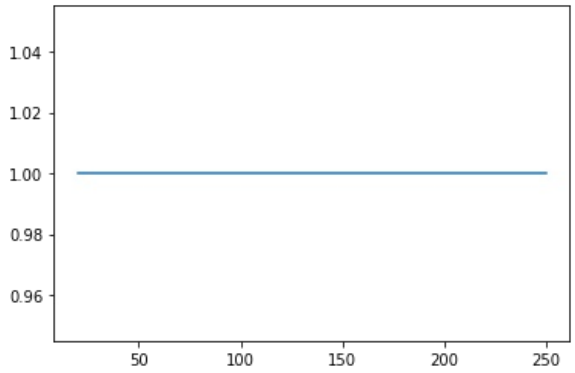
```
optimazer_4.cv_results_
y_range4=np.array(optimazer_4.cv_results_['mean_test_score'])
plt.plot(n_estimators4, y_range4)
```

Out[60]:

```
{'mean_fit_time': array([0.07072158, 0.20788922, 0.27898731, 0.45875835, 0.62413116,
        0.89777803]),
 'std_fit_time': array([0.00815477, 0.02836091, 0.02156647, 0.01436434, 0.06284819,
        0.14643006]),
 'mean_score_time': array([0.00757771, 0.01775179, 0.01935463, 0.05287743, 0.05184183,
        0.0619904 ]),
 'std_score_time': array([0.00195584, 0.00798828, 0.00149927, 0.03061406, 0.0187573 ,
        0.01983625]),
 'param_n_estimators': masked_array(data=[20, 50, 100, 150, 200, 250],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 20},
  {'n_estimators': 50},
  {'n_estimators': 100},
  {'n_estimators': 150},
  {'n_estimators': 200},
  {'n_estimators': 250}],
 'split0_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split1_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split2_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split3_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1.]),
 'mean_test_score': array([1., 1., 1., 1., 1., 1.]),
 'std_test_score': array([0., 0., 0., 0., 0., 0.]),
 'rank_test_score': array([1, 1, 1, 1, 1, 1])}
```

Out[60]:

```
[<matplotlib.lines.Line2D at 0x21a048d55c8>]
```



9.Повторение пункта 8 для найденных оптимальных значений гиперпараметров.

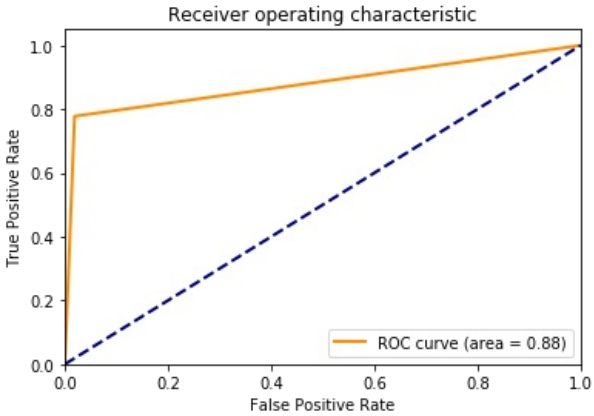
In [61]:

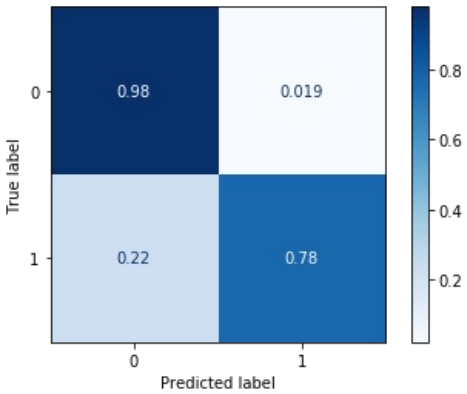
```
cl0_1 = KNeighborsClassifier(n_neighbors=optimazer_0.best_params_['n_neighbors']).fit(X_train, Y_train)
target0_1 = cl0_1.predict(X_test)
```

In [62]:

```
print('База:')
get_score('KNeighbors_base',cl0_0,target0_0)
print('Новые данные:')
get_score('KNeighbors_grid',cl0_1,target0_1)
paint(cl0_1,target0_1)
#Несмотря на то, что основным параметром для поиска оптимальных гиперпараметров был accuracy_score, такой г
иперпараметр заметно улучшил
#precision_score, но с уменьшением recall
```

```
База:
accuracy_score:  0.9504504504504504
precision_score:  0.3333333333333333
recall_score:  0.2222222222222222
Новые данные:
accuracy_score:  0.972972972972973
precision_score:  0.6363636363636364
recall_score:  0.7777777777777778
```





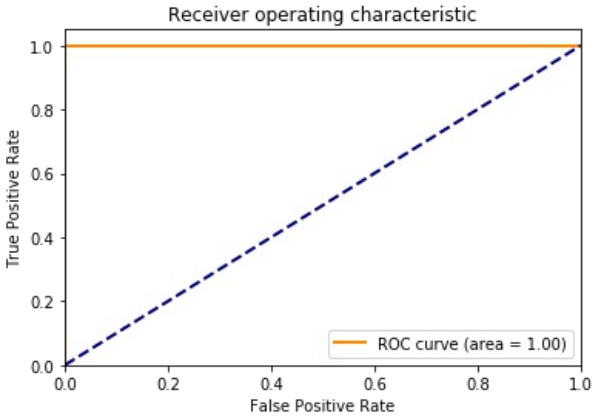
In [63]:

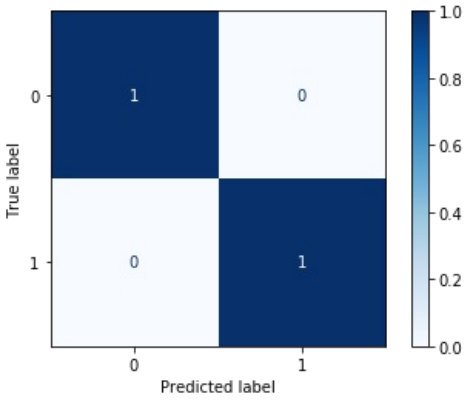
```
cl1_1 = DecisionTreeClassifier(max_depth=optimazer_1.best_params_['max_depth']).fit(X_train, Y_train)
target1_1 = cl1_1.predict(X_test)
```

In [64]:

```
print('База:')
get_score('Decision_base',cl1_0,target1_0)
print('Новые данные:')
get_score('Decision_grid',cl1_1,target1_1)
paint(cl1_1,target1_1)
#Можно заметить, насколько сильно улучшились параметры, даже несмотря на достатоно низкую (по сравнению с ближайшими соседями) скорость перебора.
```

```
База:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
Новые данные:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





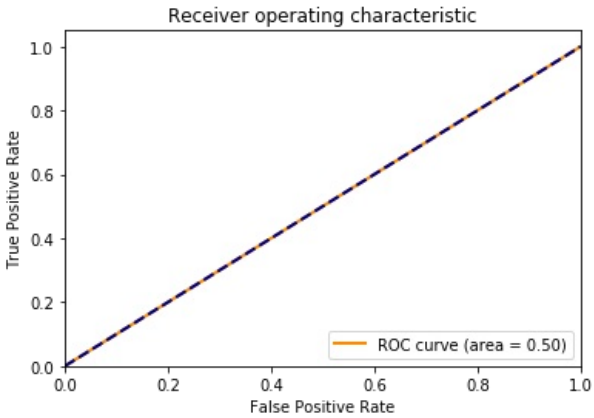
In [65]:

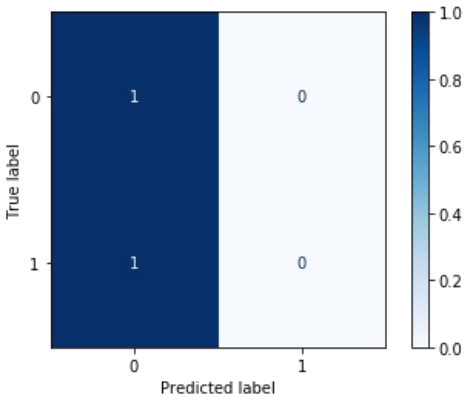
```
cl2_1 = LinearSVC(C = optimazer_2.best_params_['C']).fit(X_train, Y_train)
target2_1 = cl2_1.predict(X_test)
```

In [66]:

```
print('База:')
get_score('Linear_base',cl2_0,target2_0)
print('Новые данные:')
get_score('Linear_grid',cl2_1,target2_1)
paint(cl2_1,target2_1)
#Как мы видим, новые параметры не сильно отличаются от исходных
```

```
База:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
Новые данные:
accuracy_score:  0.9594594594594594
precision_score:  0.0
recall_score:  0.0
```





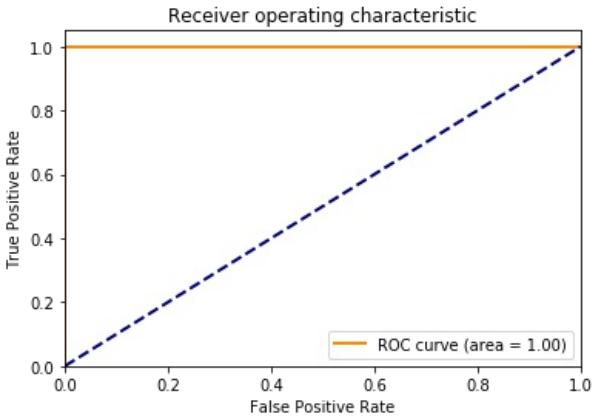
In [67]:

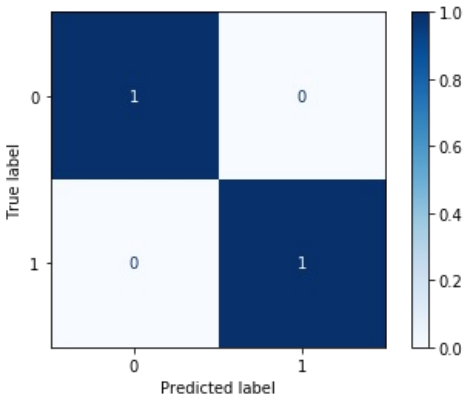
```
cl3_1 = XGBClassifier(learning_rate = optimazer_3.best_params_['learning_rate'],
                      max_depth= optimazer_3.best_params_['max_depth'],
                      n_estimators=optimazer_3.best_params_['n_estimators']).fit(X_train, Y_train)
target3_1 = cl3_1.predict(X_test)
```

In [68]:

```
print('База:')
get_score('XGBoost_base',cl3_0,target3_0)
print('Новые данные:')
get_score('XGBoost_grid',cl3_1,target3_1)
paint(cl3_1,target3_1)
#Как мы видим, новые параметры не сильно отличаются от исходных
```

```
База:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
Новые данные:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





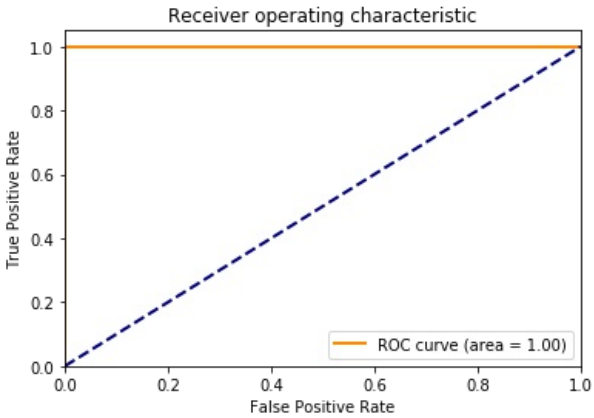
In [69]:

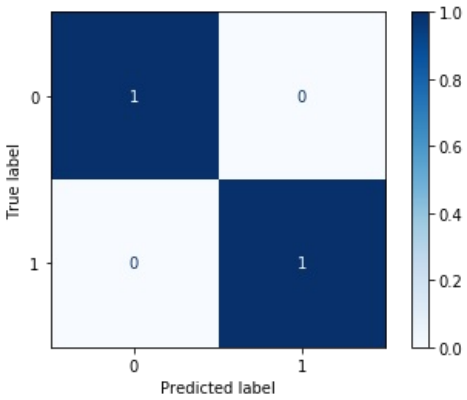
```
cl4_1 = BaggingClassifier(n_estimators= optimazer_4.best_params_['n_estimators']).fit(X_train, Y_train)
target4_1 = cl4_1.predict(X_test)
```

In [70]:

```
print('База:')
get_score('Bagging_base',cl4_0,target4_0)
print('Новые данные:')
get_score('Bagging_grid',cl4_1,target4_1)
paint(cl4_1,target4_1)
#Как мы видим, новые параметры не сильно отличаются от исходных
```

```
База:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
Новые данные:
accuracy_score:  1.0
precision_score:  1.0
recall_score:  1.0
```





10.Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.

In [71]:

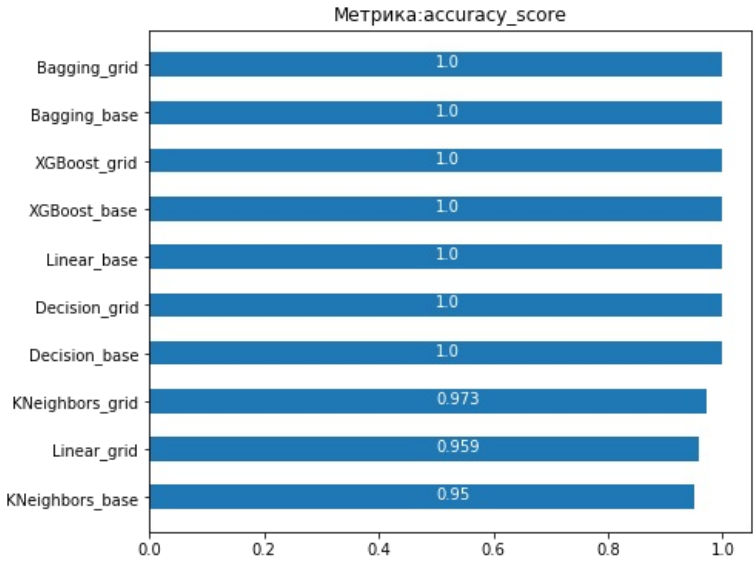
```
class_metrics = metric_logger.df['metric'].unique()
class_metrics
```

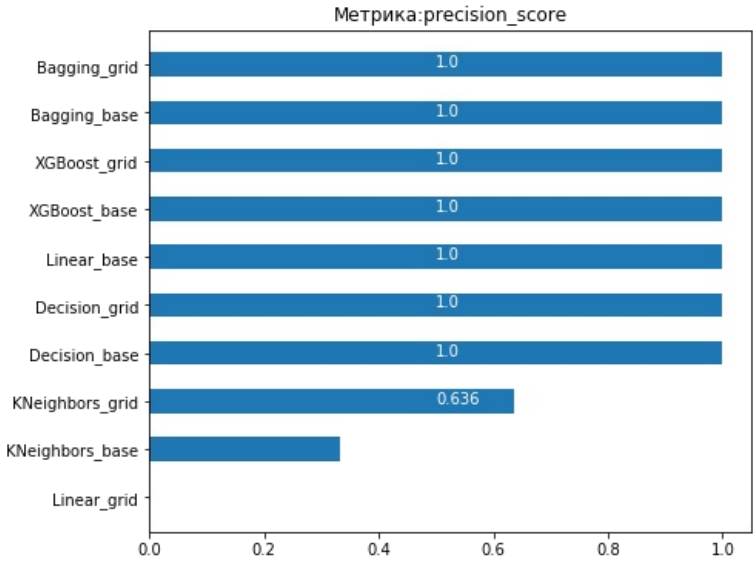
Out[71]:

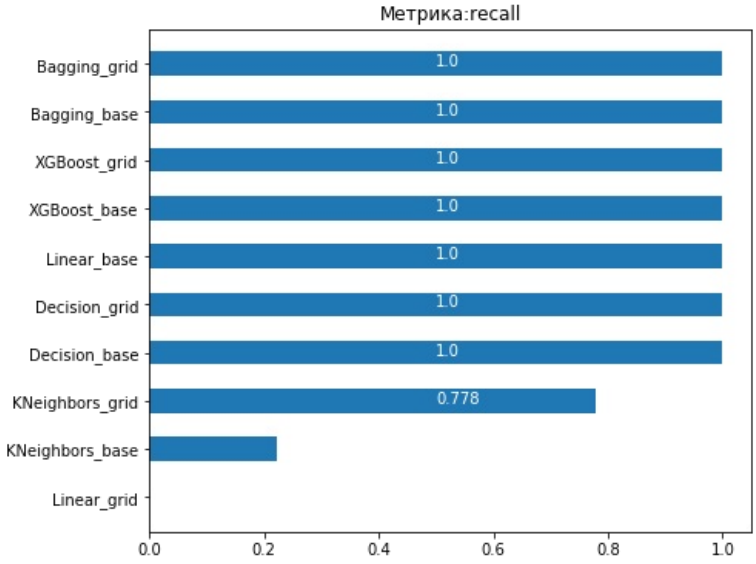
```
array(['accuracy_score', 'precision_score', 'recall'], dtype=object)
```

In [72]:

```
for metric in class_metrics:
    metric_logger.plot('Метрика:' + metric,metric, figsize=(7,6))
```







Вывод: лучшими оказались модели на основе линейной регреси.

In [ ]: